A Tutorial on Physics-Informed Neural Networks (PINNs)

From 1D ODEs to 2D PDEs, Black–Scholes, and HJB

Simon Scheidegger

Department of Economics, University of Lausanne

Thursday, March 26th, 2025

Table of Contents

What Are Physics-Informed Neural Networks (PINNs)?

- PINNs incorporate physical laws (ODEs, PDEs, etc.) into the training of neural networks.
- Instead of (or in addition to) fitting standard labeled data, we penalize the network if it violates the governing equations.
- Use automatic differentiation (AD) to compute derivatives for the PDE or ODE residual.
- Boundary/initial conditions are enforced as part of the loss function.
- We effectively transform solving differential equations into a data-driven, neural-network–based learning problem.

General PINN Strategy

1. Define a neural network $u_\theta(x)$ (or $u_\theta(x, y)$, etc. for PDEs).
2. Use automatic differentiation to compute partial derivatives needed for the PDE/ODE.
3. Define the **residual** for the equation, e.g. $F(u_\theta, x)$.
4. Incorporate boundary or initial conditions by adding penalty terms to the loss.
5. Train (optimize) to minimize:

$$\mathcal{L}(\theta) = \underbrace{\text{MSE}(F(u_\theta, x))}_{\text{Equation Residual}} + \underbrace{\text{MSE}(u_\theta(\text{boundary}) - \text{BC})}_{\text{Boundary Loss}} + \cdots$$

6. After training, u_θ approximates the solution.

1D ODE with Zero Boundary Conditions

Example ODE:

$$\frac{d^2y}{dx^2} = -1, \quad x \in (0,1),$$

with **boundary conditions**

$$y(0) = 0, \quad y(1) = 0.$$

Analytical solution:

$$y(x) = -\frac{x^2}{2} + \frac{x}{2}.$$

PINN Setup:

- Define $y_\theta(x)$ as a neural net.
- PDE residual: $r(x) = y_\theta''(x) + 1$.
- Enforce boundary conditions $y_\theta(0) = 0$ and $y_\theta(1) = 0$.

Pointers to Code

Code Reference:

- `ode_zero_bc.py` demonstrates the full PyTorch implementation.
- It constructs a small network, sets up the ODE residual, and enforces boundary conditions within the loss.
- Training is run via Adam, and a final plot compares the learned solution to the analytical solution.

Slide Note: In live presentation, you can open or run `ode_zero_bc.py` to show details.

1D ODE with Non-Zero Boundary Conditions

Example ODE (same interior PDE, different BCs):

$$y''(x) = -1, \quad x \in (0,1),$$

$$y(0) = 1, \quad y(1) = 2.$$

Analytical solution:

$$y(x) = -\frac{x^2}{2} + \frac{x}{2} + 1.$$

Pointers to Code

Main Difference:

- The boundary conditions now are $y(0) = 1$ and $y(1) = 2$.
- Only the boundary part of the loss changes.

Code Reference:

- `ode_nonzero_bc.py` shows the minor modifications in the boundary loss terms.

Example: 2D Laplace with Zero BCs

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad (x, y) \in (0,1) \times (0,1),$$

with boundary $u = 0$ on $\partial\Omega$.
Trivial solution: $u(x, y) = 0$.
PINN approach:

- $u_\theta(x, y)$ as a neural net.
- Residual: $u_{xx} + u_{yy} = 0$.
- Enforce $u = 0$ on all edges.

Pointers to Code

Code Reference:

- `laplace_2d_zero_bc.py`.
- Implements interior sampling for (x, y) and boundary sampling on the edges.
- Laplace residual computed via second derivatives from ‘torch.autograd.grad‘.

Example: 2D Laplace with Non-Zero BCs

$$u_{xx} + u_{yy} = 0, \quad (x, y) \in (0, 1) \times (0, 1),$$

with $u(0, y) = 0$, $u(1, y) = 1$, $u(x, 0) = 0$, $u(x, 1) = 1$.
Analytical solution: $u(x, y) = \frac{x+y}{2}$.
Key difference:

- The boundary condition is no longer zero but $\{0, 1\}$.
- Adjust boundary losses accordingly.

Pointers to Code

Code Reference:

- `laplace_2d_nonzero_bc.py`.
- Shows how to define a boundary function for each edge and enforce it in the boundary part of the loss.

Black–Scholes Recap

$$\frac{\partial V}{\partial t} + \tfrac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.$$

$$\text{Terminal: } V(S,T) = \max(S-K,0).$$

$$\text{Boundaries: } V(0,t) = 0, \quad V(S_{\max},t) \approx S_{\max} - Ke^{-r(T-t)}.$$

Pointers to Code

Key Steps in the PINN Setup:

- PDE residual:

$$V_t + 0.5\,\sigma^2 S^2\, V_{SS} + rS\, V_S - r\, V\,.$$

- Boundary conditions at $S = 0$ and $S = S_{\text{max}}$.
- Terminal condition at $t = T$.

Code Reference:

- `black_scholes_pinn.py` contains the full implementation.

HJB: Problem Setup

We consider a 1D Hamilton–Jacobi–Bellman (HJB) PDE:

$$-r\,V(x) \;+\; \sup_{a\in[-1,1]} \Big[(x+a)\,V'(x) \;-\; \alpha\,a^2 \Big] = 0, \quad x \in (0,1).$$

Boundary conditions:

$$V(0) = 0, \quad V(1) = 1.$$

Interpretation:

- $r > 0$ is a discount rate.
- $\alpha > 0$ penalizes the square of the control a.
- The state variable is $x \in [0,1]$.

Optimal Control:

$$a^*(x) \;=\; \mathrm{clamp}\Bigl(\frac{V'(x)}{2\,\alpha}\,,\;-1,\;1\Bigr).$$

Then the PDE becomes:

$$-r\,V(x) \;+\; (x + a^*)\,V'(x) \;-\; \alpha\,(a^*)^2 \;=\; 0.$$

Key point: If $|\,V'(x)/(2\alpha)\,| \le 1$, the unconstrained optimum is valid. Otherwise, it saturates at ± 1.

PINN Formulation

- Let $V_\theta(x)$ be a fully connected neural network with parameters θ.
- We compute $V'(x)$ via **automatic differentiation**.
- PDE residual:

$$R_\theta(x) \;=\; -r\,V_\theta(x) \;+\; \big(x + a^*(x)\big)\,V_\theta'(x) \;-\; \alpha\,\big(a^*(x)\big)^2.$$

- **Loss** for interior points:

$$\mathcal{L}_{\mathrm{PDE}}(\theta) = \sum_{x_{\mathrm{int}}} \big(R_\theta(x_{\mathrm{int}})\big)^2.$$

- **Boundary conditions**:

$$V_\theta(0) = 0, \quad V_\theta(1) = 1 \quad \Rightarrow \quad \mathcal{L}_{\mathrm{BC}}(\theta) = \big(V_\theta(0) - 0\big)^2 + \big(V_\theta(1) - 1\big)^2.$$

- **Total loss**: $\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{BC}}$.

Implementation

- **Network architecture**: fully-connected, Tanh activations.
- **Optimization**: Adam with a moderate learning rate (e.g., 10^{-3}).
- **Sampling**:
  - **Interior points**: random $x \in (0, 1)$.
  - **Boundary points**: $x = 0$, $x = 1$.
- **Code Reference**:
  - See `hjb_pinn_continuous.py` for a complete example.
  - After training, we plot $V_\theta(x)$ on $x \in [0, 1]$.

Results

- The PINN learns $V(x)$ that (approximately) satisfies the HJB PDE and boundary conditions.
- The learned policy is $a^*(x) = \mathrm{clamp}(V'(x)/(2\alpha), -1, 1)$.
- No closed-form analytical solution is needed; we compare the training residual or do a quick check of boundary conditions.

Conclusion: PINNs can handle continuous controls in HJB equations by *differentiating* w.r.t. $V_\theta(x)$, computing the *optimal control*, and enforcing the PDE residual.

PDE Setup with non-trivial Boundary Conditions

We consider a 2D HJB:

$$-r\,V(x,y) \;+\; \max_{a\in[-1,1]}\Big[(x+a)\,\frac{\partial V}{\partial x} \;+\; (y+a)\,\frac{\partial V}{\partial y} \;-\; \alpha\,a^2\Big] = 0,$$

with $(x,y) \in (0,1)\times(0,1)$.
Non-trivial boundary conditions:

$$V(0,y) = y, \quad V(1,y) = 1+y, \quad V(x,0) = x, \quad V(x,1) = x+1.$$

- $r > 0$ is discount factor, $\alpha > 0$ is penalty on a^2.
- PDE domain: the unit square.

Continuous Control in [-1,1]

Optimal control $a^*(x, y)$:

$$a_{\text{unclamped}} = \frac{\frac{\partial V}{\partial x} + \frac{\partial V}{\partial y}}{2\,\alpha}, \quad a^* = \mathrm{clamp}(a_{\text{unclamped}}, -1, 1).$$

Then PDE residual:

$$R(x, y) = -r\,V + (x + a^*)\,\frac{\partial V}{\partial x} + (y + a^*)\,\frac{\partial V}{\partial y} - \alpha\,(a^*)^2 = 0.$$

PINN Formulation

Neural Network: $V_\theta(x, y)$.

- **Interior loss**:

$$\mathcal{L}_{\mathrm{PDE}}(\theta) = \sum_{(x_i, y_i) \in \text{interior}} \big(R_\theta(x_i, y_i)\big)^2.$$

- **Boundary loss**:

$$\mathcal{L}_{\mathrm{BC}}(\theta) = \sum_{(x_j, y_j) \in \text{boundary}} \Big(V_\theta(x_j, y_j) - \mathrm{BC}(x_j, y_j)\Big)^2,$$

  where $\mathrm{BC}(x, y)$ is the piecewise function that returns $y, 1 + y, x, x + 1$ on each edge.

- **Total loss**:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathrm{PDE}} + \mathcal{L}_{\mathrm{BC}}.$$

Implementation

- **Network**: fully connected, e.g., Tanh activation, 64 hidden units.
- **Sampling**:
  - $\mathrm{N}_{\mathrm{interior}}$ random points in $(0, 1) \times (0, 1)$.
  - $\mathrm{N}_{\mathrm{boundary}}$ random points on each edge.
- **Optimizer**: Adam, with moderate LR (e.g. 10^{-3}).
- **Loss printing**: PDE loss, BC loss, total loss each iteration (or every 500 epochs).

Code Reference:

- See `hjb_2d_nontrivial_bc.py` for the full script.
- After training, we plot V_θ in a 2D color map and a 3D surface.

Summary 2D HJB with non-trivial BCs

- We introduced a 2D HJB PDE with non-trivial BCs.
- We clamp the control $a^* = \mathrm{clamp}(\frac{V_x+V_y}{2\alpha}, -1, 1)$ inside the PDE.
- The PINN approach adds PDE residual loss + boundary mismatch loss.
- This yields $V_\theta(x, y)$ that approximates the PDE solution subject to the specified boundary condition.

Summary & Next Steps

- We introduced PINNs: enforcing PDE/ODE constraints by building them into the loss function.
- Showed ODEs, PDEs, boundary conditions, and terminal conditions.
- Demonstrated examples: 1D boundary value problems, 2D Laplace, Black–Scholes, and a simple HJB.
- In practice, you can adapt these templates to more complex PDEs, domains, or multi-dimensional states.

References & Further Reading

- M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-Informed Neural Networks: A Deep Learning Framework (2019).
- J. Berg and K. Nystroem, A Unified Deep Artificial Neural Network Approach to PDEs in Complex Geometries (2017).
- For HJB PDE references in reinforcement-learning contexts, see: D. Jiang, F. Meng, Q. Sun, X. Xue, and Y. Zou. DeepRitz Method (2019).

Thank You!

Questions?